ИЗПИТ ПО ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ ЗА СПЕЦИАЛНОСТ “КОМПЮТЪРНИ НАУКИ” — 2. КУРС, 1. ПОТОК (СУ, ФМИ, 17 ЮНИ 2021 Г.)

Указания:

1) Имената на всички алгоритми и структури от данни да са на български език.
2) Всички предложени алгоритми да бъдат обосновани и анализирани по време.

Не се приемат решения, в които е нарушено някое от горните изисквания!

Задача 1. Внучка наследява от баба си винарска изба с n бутилки вино, наредени в редица и номерирани от 1 до n. Началните им цени са реални неотрицателни числа, дадени в масива $P[1 \ldots n]$. Колкото по-дълго отлежават бутилките, толкова по-скъпи стават: цената на бутилка № k след x години ще бъде $x \,.\, P[k]$.

В завещанието си бабата е поискала всяка година внучката да продава по една бутилка, като избира или най-лявата, или най-дясната останала. Каква е максималната парична сума, която внучката може да спечели и в какъв ред трябва да продава бутилките? Смятаме, че бутилките са отлежали една година, когато се продава първата от тях.

Предложете алгоритъм с полиномиална времева сложност, който намира най-голямата възможна печалба. Опишете алгоритъма на псевдокод. **1 точка**

Разширете алгоритъма така, че да намира и оптималната последователност за продажба на бутилките. Опишете алгоритъма на псевдокод. **1 точка**

Докажете коректността и пресметнете времевите сложности на алгоритмите.

Задача 2. Дадени са низовете $A[1 \ldots n]$ и $B[1 \ldots m]$, където $n > m > 0$. Съставете алгоритъм за проверка дали $B[1 \ldots m]$ е подредица на $A[1 \ldots n]$. Алгоритъмът трябва да работи за време $O(n)$ при всякакви входни данни. Опишете алгоритъма на псевдокод. Обосновете коректността му. Докажете, че времевата му сложност изпълнява поставеното изискване. **1 точка**

Задача 3. Във всеки връх на пълен неориентиран граф стои реално число. Теглото на всяко ребро е абсолютната стойност на разликата на числата в краищата му. Съставете алгоритъм, който по дадени числа на върховете намира минимално покриващо дърво за време $o(n^2)$ при всякакви входни данни, където n е броят на върховете. Опишете алгоритъма с думи и дайте пример. Обосновете коректността му и пресметнете времевата му сложност. **1 точка**

СХЕМА ЗА ОЦЕНЯВАНЕ

Всяка точка се дава само при изпълнение на всички поставени изисквания. Оценката = 2 + t, където t е броят на получените точки.

РЕШЕНИЯ

Задача 1 се решава с ***динамично програмиране.*** Отговаряме на въпросите: "Колко най-много може да спечели внучката, ако разполага само с бутилките от № *left* до № *right* включително?" Тук $1 \le left \le right \le n$. Пазим отговорите в клетките *dyn*[*left*, *right*] на триъгълна динамична таблица. Можем да пазим и избора на бутилка за продажба (най-лявата или най-дясната), като използваме логическа триъгълна таблица *taken*; по-точно, *taken*[*left*, *right*] = истина ⇔ най-голяма печалба се постига с продажбата на най-лявата бутилка.

```
maxWine(P[1...n]: array of real numbers): real number
dyn[1...n, 1...n]: array of real numbers; // max печалба
taken[1...n, 1...n]: array of boolean; // true <=> left
year ← n
for left ← 1 to n do
   dyn[left, left] ← year × P[left]  // right = left
for year ← n-1 downto 1 do
   for left ← 1 to year do
      right ← left + n - year
      winL ← year × P[left]  + dyn[left + 1, right]
      winR ← year × P[right] + dyn[left, right - 1]
      if winL > winR  // продава се най-лявата бутилка
         dyn[left, right] ← winL
         taken[left, right] ← true
      else  // продава се най-дясната бутилка
         dyn[left, right] ← winR
         taken[left, right] ← false
left ← 1
right ← n
while left ≤ right do // taken[left, left] няма значение,
   if taken[left, right]   // затова не е инициализирано
      print "Продай най-лявата бутилка."
      left ← left + 1
   else
      print "Продай най-дясната бутилка."
      right ← right - 1
return dyn[1, n]
```

Времевата сложност на алгоритъма е $\Theta\left(n^2\right)$ заради двата вложени цикъла: другите два цикъла отнемат пренебрежимо малко време — само O(n).

Коректността на стойностите в динамичната таблица $dyn[1\ldots n,\ 1\ldots n]$ (таблицата е триъгълна, защото първият индекс е не по-голям от втория) се доказва чрез математическа индукция по разликата $right - left$:

— База: при $right = left$ (разлика 0). Тогава редицата съдържа една бутилка. Това се случва в последната година — № n. Затова цената на тази бутилка трябва да се умножи по n, както прави първият цикъл от алгоритъма, попълващ диагонала на динамичната таблица.

— Индуктивна стъпка: при $right > left$ (положителна разлика). Сега редицата съдържа поне две бутилки, затова внучката има две възможности: да продаде или най-лявата, или най-дясната бутилка. Най-голямата възможна печалба в тези случаи (при оптимални следващи продажби) е съответно *winL* и *winR*, а общо — по-голямата от двете печалби. Сравнението между тях се извършва от условния оператор във вложените цикли. Двете присвоявания преди него са коректни, защото първоначалната цена на продадената бутилка се умножава по броя на изтеклите години, а най-голямата възможна печалба оттук нататък е пресметната и записана съответно в $dyn[left+1,\ right]$ и $dyn[left,\ right-1]$. Това следва от индуктивното предположение, защото разликата на индексите на тези две клетки от динамичната таблица е с единица по-малка от разликата на индексите на клетката, която алгоритъмът попълва в момента:

$$right - (left + 1)\ =\ (right - 1) - left\ =\ (right - left) - 1.$$

Индуктивното доказателство е завършено.

С горните разсъждения всъщност доказахме следната рекурентна формула:

$$dyn[left, right] = \begin{cases} n\ .\ P[left] \text{ при } right = left; \\ \max\begin{cases} (n - right + left)\ .\ P[left]\ +\ dyn[left+1, right] \\ (n - right + left)\ .\ P[right] + dyn[left, right-1] \end{cases} \text{ при } right > left. \end{cases}$$

Тази формула е програмирана в описания алгоритъм.

Алгоритъмът все някога приключва работа. За циклите по брояч това е ясно. Че цикълът по условие в края на алгоритъма също приключва рано или късно, следва от полуинварианта $right - left$: това е цяло число, намаляващо с единица при всяко изпълнение на тялото на цикъла, ето защо тази разлика все някога ще стане отрицателна и цикълът ще завърши.

Накрая алгоритъмът връща $dyn[1,\ n]$, тоест най-голямата възможна печалба, постижима, когато внучката разполага с бутилките от № 1 до № n включително (тоест с всички бутилки). Това е точно отговорът на задачата.

Резултатът от всяко сравнение между *winL* и *winR* се пази в масива *taken*, затова, тръгвайки от $taken[1,\ n]$, намираме най-добра редица от продажби.

Условието на задачата и идеята на решението са от учебните материали на www.informatika.bg.

Задача 2. Алгоритъмът проверява първо дали $B[1]$ се среща в $A[1 \ldots n]$. Ако не се среща, то $B[1 \ldots m]$ не е подредица на $A[1 \ldots n]$; край на алгоритъма. Ако $A[1 \ldots n]$ съдържа $B[1]$, алгоритъмът запомня първото такова място k_1. По-нататък алгоритъмът проверява дали знакът $B[2]$ се среща в $A[k_1+1 \ldots n]$. Ако не се среща, то $B[1 \ldots m]$ не е подредица на $A[1 \ldots n]$; край на алгоритъма. Ако $A[k_1+1 \ldots n]$ съдържа $B[2]$, алгоритъмът запомня първото такова място k_2. След това алгоритъмът проверява дали знакът $B[3]$ се среща в $A[k_2+1 \ldots n]$. Ако не се среща, то $B[1 \ldots m]$ не е подредица на $A[1 \ldots n]$; край на алгоритъма. Ако $B[3]$ се среща в $A[k_2+1 \ldots n]$, алгоритъмът запомня първото такова място k_3. И тъй нататък. Ако намери и знака $B[m]$, то $B[1 \ldots m]$ е подредица на $A[1 \ldots n]$.

```
ALG(A[1...n], B[1...m])
1) q ← 1
2) for k ← 1 to n do
3)    if A[k] = B[q]
4)       q ← q + 1
5)       if q > m
6)          return true
7) return false
```

Заради цикъла с начало на ред № 2 от кода времевата сложност е $\Theta(n)$ при всякакви входни данни.

Това е ***алчен алгоритъм***, защото взима първото място в A, където успее да намери знака $B[q]$. Ще докажем, че алгоритъмът е коректен; с други думи, ако $B[1 \ldots m]$ изобщо е подредица на $A[1 \ldots n]$, то тя може да бъде намерена по описания начин. Наистина, щом низът $B[1 \ldots m]$ е подредица на $A[1 \ldots n]$, то $k_1 < k_2 < \ldots < k_m$ според определението за подредица, където $A[k_q] = B[q]$. Нека $\widetilde{k}_1$ е най-малкият индекс, за който $A[\widetilde{k}_1] = B[1]$. Поради минималността му е изпълнено неравенството $\widetilde{k}_1 \le k_1$. Ето защо $\widetilde{k}_1 < k_2 < \ldots < k_m$, тоест отново имаме подредица на $A[1 \ldots n]$, но сега за знака $B[1]$ е взето първото му срещане в низа $A[1 \ldots n]$. Затова алгоритъмът не греши, взимайки първото срещане $\widetilde{k}_1$ на $B[1]$ в $A[1 \ldots n]$. От неравенствата $\widetilde{k}_1 < k_2 < \ldots < k_m$ следва, че $B[2 \ldots m]$ е подредица на $A[\widetilde{k}_1 + 1 \ldots n]$, затова алгоритъмът продължава да увеличава индекса k. Търсенето на подредица на $A[\widetilde{k}_1 + 1 \ldots n]$, съвпадаща с $B[2 \ldots m]$, е същата задача, но с други входни данни. Затова важи същото разсъждение: алгоритъмът няма да сбърка, като вземе първото срещане в низа $A[\widetilde{k}_1 + 1 \ldots n]$ на първия знак от новия подниз на B, тоест $B[2]$. Същото се отнася за $B[3]$ и т.н.

Задача 3. Минималното покриващо дърво на граф от описания вид е път, при движение по който числата на върховете само растат или само намаляват (допуска се да остават същите). Тоест достатъчно е да сортираме върховете, а това може да стане за време $\Theta(n \log n) = o(n^2)$ при всякакви входни данни, например с помощта на пирамидално ***сортиране.***

Действително, нека a и b са съответно най-голямото и най-малкото число, написани във върховете на графа. Всяко покриващо дърво е свързан граф, затова съдържа път между числата a и b. Теглото на пътя е равно на сбора от абсолютните стойности на разликите на числата в последователните върхове от пътя, а според неравенството на многоъгълника въпросният сбор е по-голям или равен на $a - b$. Понеже теглата на всички ребра на графа са неотрицателни, то теглото на покриващото дърво е поне $a - b$. Този извод важи за произволно покриващо дърво на графа.

От друга страна, пътят, преминаващ през всички върхове в ненарастващ ред на числата в тях, е покриващо дърво с тегло

$$(a - c) + (c - d) + (d - f) + \ldots + (g - h) + (h - b) = a - b,$$

защото всички събираеми се унищожават с изключение на a и b. Тук

$$a \geq c \geq d \geq f \geq \ldots \geq g \geq h \geq b$$

са последователните върхове от разглеждания път, тоест той има следния вид:

$$a \text{ — } c \text{ — } d \text{ — } f \text{ — } \cdots \text{ — } g \text{ — } h \text{ — } b.$$

Този път е покриващо дърво, защото преминава по веднъж през всеки връх, а представлява минимално покриващо дърво, защото теглото му е точно $a - b$, докато теглото на всяко друго покриващо дърво е поне $a - b$.